CHAIR OF DECENTRALIZED INFORMATION SYSTEMS & DATA MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Thesis proposal

Concurrent Range Locking

Author: **Thua Duc Nguyen**

Supervisor: **Prof. Dr. Viktor Leis**

Advisor: **Lam Duy Nguyen**

1 Introduction

A concurrent range lock is a dynamic fine-grained lock abstraction. It is designed to address the problem of multiple writers attempting to modify different parts of the same file. Unlike conventional methods that rely on a singular lock for the entire file, range lock enables concurrent access to disjoint parts of the file, thus allowing parallelism for writers and solving the bottleneck of using a singular lock.

2 Motivation

Recently, there has been an increase in interest in range locking techniques. One notable example is that the Linux kernel community is considering using range-locking techniques to replace the `mmap_lock` [1, 2, 3]. The `mmap_lock` uses a per-process semaphore to control access to the whole `mm_struct` [4] and serialize changes to address spaces. There have been attempts to overcome the scalability issues of `mmap_lock`, but the problem is far from solved [3].

In database management systems context, range locks also offer a solution to the issue of coarse-grained locking in large databases and indexes. Instead of locking entire indexes, range locks focus on key ranges, reducing contention and improving concurrency. This allows multiple transactions to operate concurrently on separate key ranges, alleviating memory overhead and lock acquisition bottlenecks [5].

3 Related Work

Previous research has explored various approaches to scalable range lock. The current implementation in the Linux kernel uses an interval tree [6], which keeps track of the ranges that have been acquired and requested, and an internal spin lock to protect it. This spin lock becomes a point of contention on its own when the range lock is frequently acquired.

A similar range lock version of the Linux kernel implementation uses a skip list combined with a spin lock to dynamically maintain currently locked ranges [7]. The skip list is lighter than the interval tree and still sufficient for intensive searches for existing overlapping ranges. But the contention point issue is still there.

One different approach to range lock is to build it based on a linked list, where each node represents an acquired range [8]. Although using a list makes it possible to maintain a lock-free range lock, insertion and lookup operation on the linked list are likely to be less efficient.

4 Approach

The proposed approach is to design and implement a concurrent range lock based on a probabilistic concurrent skip list. Unlike the conventional skip list, the proposed approach would use the requesting range `[start, end]` as the key per node and. It would also use the per-node lock technique. This method will not only solve the bottleneck problem of the spin lock base range lock, but also keep the performance of the lock high.

5 Evaluation

The proposed approach will be evaluated under these evaluation criteria:

- Performance: It is necessary to measure the performance and throughput of the range locking mechanism under increasing load and concurrent accesses.
- Correctness: It is also crucial to validate the consistency and correctness of data accesses, especially in scenarios involving overlapping data ranges and concurrent operations.
- Comparison: A comparison must be made between the performance of the proposed solution and those of existing state-of-the-art approaches.

6 Expected Outcome

The expected outcome of this research is a scalable range-locking mechanism that improves the efficiency of the existing range locks. The findings from the evaluation will provide insights into the performance characteristics, and potential trade-offs of the proposed mechanism, contributing to the advancement of distributed computing research.

7 Resources

A thorough evaluation of the proposed mechanism requires a budget of 32 cores and 32 GB of RAM for one month. These resources allow for exhaustive testing under heavy contention, pushing the range lock to its limits and revealing its performance in multithreaded scenarios.

Bibliography

[1] J. Corbet. "Range reader/writer locks for the kernel". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/724502/.

[2] M. Rybczynska. "Introducing maple trees". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/845507/.

[3] J. Corbet. "The ongoing search for mmap_lock scalability". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/893906/.

[4] S. Boutnaru. "Linux Kernel — mm_struct". In: *medium.com* (2023). Accessed: 2024-04-21. URL: https://medium.com/@boutnaru/linux-kernel-mm-struct-fafe50b57837.

[5] G. Graefe. *Hierarchical Locking in B-Tree Indexes*. Accessed: 2024-04-21. 2020. URL: https://dl.gi.de/server/api/core/bitstreams/9f283897-1a24-4714-aa57-c773e6c2da96/content.

[6] J. Kara. "Implement range locks". In: *lkml.org* (2013). Accessed: 2024-04-21. URL: https://lkml.org/lkml/2013.

[7] X. Song, J. Shi, R. Liu, Y. Jian, and H. Chen. *Parallelizing Live Migration of Virtual Machines*. Accessed: 2024-04-21. 2013. URL: https://ipads.se.sjtu.edu.cn/_media/publications:parallel-migration.pdf.

[8] A. Kogan, D. Dice, and S. Issa. *Scalable range locks for scalable address spaces and beyond*. Accessed: 2024-04-21. 2020. URL: https://arxiv.org/pdf/2006.12144.pdf.